DB2 PROJECT 1 DESIGN REPORT

RIGOROUS 2PL & WOUND WAIT METHOD

Nikitha Chennamaneni -1001745322

Project Description

The implementation of this will make use of HashMap class of JAVA for storing and processing the transactions and the locks applied on the data items.

Language used:

Java

Data Structures:

The implementation of the simulation will make use of HashMap class of JAVA for storing and processing the transactions and the locks applied on the data items.

The following are the table structures that will be used to capture the transaction and locking information:

Transaction table: It contains details such as transaction id, transaction time stamp, transaction state, a list of data items held in FIFO order.

The data structure used is Hash table**.**

Key – Transaction Id

Value – Transaction object – This contains the details about the transaction such as:

- String transactionState : This is used to store the status of the transaction like 'Active', 'Blocked', 'Aborted', 'Committed'
- int transactionId : To store the Transaction ID of aparticular transaction
- int timeStamp: To store the Time stamp
- List itemsHeld = new ArrayList() This is the store the items held by the transaction.
- Queue of Operation objects waitingOperations using linked list;

Lock table: It contains details such as data item name, type of lock, list of transactions accessing the item and a list of waiting transactions.

The data structure used is Hash table**.**

Key – Data item

Value – Lock object - This contains the details such as:

- String itemName: This stores the Data Item
- String lockState: This is the store operation like ‘read’ or ‘write
- PriorityQueue<String> readLockTransId to hold the read locked transaction ids

  String writeLockTransId to hold the write locked transaction id.
- List<Integer waitingTID = new ArrayList<Integer>(): To store Waiting Transation IDs >

Implementation:

Step 1. Read from file

The program starts from Main method and the input file is read line by line. As we read a line in the in the input file we will store Transaction Id(TID), operation(begin(b),read(r),write(w)..), data item.

The program consists of a switch case which calls methods for begin transaction - ‘b’, read operation - ‘r’, write operation - ‘w’ and end transaction - ‘e’.

Pseudocode:

```
Main ()
{
        readInput()
        {
                1.read input file line by line

                2.if  line starts with “b” then
                        call beginTransaction(Transaction id, Time stamp, “ACTIVE”)

                3.if  line starts with “r” then
                        call readTransaction (Transaction id, data item)

                4.if  line starts with “w” then
                        call writeTransaction (Transaction id, data item)

                5.if line starts with “e” then
                        call end or abortTransaction (Transaction id, “COMMIT”)
        }
}
```

Step 2. Begin Transaction

Class TransactionDetails uses HashMap- TransactionTable is used to store the following information: Transaction id, transaction timestamp, transaction state and List of items locked (Lists of Items held).

Status will be 'Active' for all new transactions. Timestamp will be incremented whenever a new transaction is read from input file. Initially set to 1, store the record in hashmap and display it.

Pseudocode:

beginTransaction (Transaction id)

{

1. Create a entry in the transaction table(HashMap) with key equals Transaction id and value is transaction object which stores the Timestamp and Transaction State and List of Items held.
2. Update the variables of the transaction object i.e, state is updated as "Active" and Timestamp is incremented by 1
3. Create Arraylists for storing list of data items held and list of waiting operations respectively.
4. The list of data items held is stored in HashSet and the list of and waiting operations are stored in the queue

}

Step 3.Read transaction function:

This Read function is called when a read operation - 'r' is encountered from the input file. The input to this fuction is the corresponding Transaction id and data item.

If the input file reads 'r' then

read function is called and read_lock on item is requested

Pseudocode:

readTransaction(TransactionId, dataItem)

{

If trans_State is equal to "Active"

If lock table contains the ItemName

Transaction in the lock table is updated to read-locked

If the item is not locked

An entry is created in the lock datastructure with, the lock state is set to read and the transaction id and itemName are read and displayed

End If;

Else if the item is already locked by a read operation/read-write conflict

readReadfuction is called for "Read Operation" and writeReadfuction is called for "Read-Write Conflict" by using wound wait prevention protocol respectively.

End Else;

End If;

Else

Create new lock with itemName and lock State, add the transaction id and store it in the lock table

End Else;

End If;

Else

if transaction is blocked

If lock table doesnot contains the ItemName

The lock state is set to write and the transaction id and itemName are read and displayedtransaction table. These operations will be executed once operation is resumed

End if

End If;

Else (Aborted)

Unlock any items that are locked and restart again with the same timestamp;

End Else;

End Else

}

Step 4. Write transaction function

If the input file reads ‘w’ then

‘write’ function is called then a write_lock on item is requested

Pseudocode:

```
writeTransaction(transactionId, dataItem){
    If  transactionState is equal to "Active"
        If lock table contains the ItemName
            If the lockstate of the item is empty
                Lock state of the item is set to write and the item is added to
                the list of items held by the transaction
            Endif
            Else
                if the item is already read locked/read write conflict
                    readWrite function is called to resolve the conflictby
                using wound wait prevention protocol.
                End if
                Else if the item is already read locked/read write conflict
                writeWritefunction is called to resolve the conflictby using
                wound wait prevention protocol.
                End else
            End else
        End if
        Else
            Create new lock with itemName and lock State, add the transaction id
            and store it in the lock table
        End else
    End if
    Else
```

If the transaction state is blocked

if lock table doesnot contains the ItemName

Create an entry in the lck table with item name, lock state,transactio id as null.

End if

Transaction state is changed to 'blocked' and addoperations to thepriority queue in order that arewaiting to be executed in the transaction table. Theseoperations will be executed once operation isresumed

End if

Else if(Aborted)

Unlock any items that are locked and restart again with the same timestamp;

End Else;

End else

}

Step 5: End(commit) or Abort Transaction Method:

This method implements the logic to abort a transaction or commit a transaction.

If the input file reads 'e' then

abortTransaction method is called

Pseudocode:

abortTransaction(transactionId){

Item name, transaction id and transaction state are retrieved

If transaction state is "Active"

Set transaction to commit and release all the locks for items held by the transaction.

End if

Else

if transaction state is "Blocked"

Blocked function is called

End if

Else transaction state is “Abort”

Transaction is aborted

End else

End else}

Wound Wait method:

It compares the timestamp of the requested transaction with the one that has a lock on the data item.

Pseudocode:

woundWait(itemName,reqTransaction,heldTransaction, lock){

if reqTransaction.timeStamp less than heldTransaction.timeStamp

held transaction is aborted

if reqTransaction operation is read

set the lock state of the item to read and add transaction id to the item entry in lock table

end if

else reqTransaction operation is write

set the lock state of the item to write and add transaction id to the item entry in lock table

end else

end if

else

The request transaction state is said to block

end else

}

Wait-Die:

Even wait-die compares the timestamp of the requested transaction with the one that has a lock on the data item.

```
WaitDie(dataitem,reqTransaction,heldTransaction, lock){
    if reqTransaction.timeStamp greater than heldTransaction.timeStamp
        reqtransaction is aborted
        if heldTransaction operation is read
            set the lock state of the item to read and add transaction id to the
            item entry in lock table
        end if
        else heldTransaction operation is write
            set the lock state of the item to write and add transaction id to the
            item entry in lock table
        end else
    end if
    else
        The request transaction state is said to blocked
    end else
}
```

Cautious Waiting:

```
cautiousWaiting(dataitem,reqTransaction,heldTransaction, lock){
    if lockitem state is read
        place all the readlocktrans ids in the holdingTransids list
    elseif lockitem state is write
        place all the writelocktrans ids in the holdingTransids list
```

store status of all holding transaction in a list “Status”

if status list contains a blocked transaction

reqtransaction is aborted

end if

else

The request transaction state is said to blocked

end else

}

Execution Instructions:

This project is implemented using java. Main method is present in TwoPLProtocol.java with Wait-Die and Cautious Waiting methods

Compilation of java files:

Navigate to the code folder in command prompt

javac *.java

Execution:

java TwoPLProtocol

Enter the input file path when prompted

Note: The final input and output files for both Wait-Die and Cautious Waiting methods are placed in “wait-die” and “cautious-Waiting” folders